WINDOWS BASED BLOCK DIAGRAM SIMULATION FOR DSP SYSTEMS : I -- IMPLEMENTATION OF OBJECT WINDOWS INTERFACE

A thesis submitted
in partial fulfillment of the requirements
for the degree of

Master of Technology

by

Major V P Sharma

to the

Department of Electrical Engineering
Indian Institute of Technology , Kanpur
March 1995

EE- 1995- M- SHA- WIN

CERTIFICATE

It is certified that the work outlined in this thesis titled **Windows Based Block Diagram Simulation for DSP Systems :I - Implementation of Object Windows Interface** by Major V P Sharma has been carried out under my supervision and that thi work has not been submitted elsewhere for a degree .

Anil Mahanta

Dr. Anil Mahanta
Dept. of EE
IIT Kanpur

March 1995

ABSTRACT

In this thesis , a Windows based package for block diagram simulation of DSP systems has been developed . It is intended to serve as a training aid for conceptualization and experimentation of DSP systems . It can be viewed as a software breadboard which offers a flexible environment to rapidly test and verify the efficacy of a particular design using discrete functional building blocks .The software has been simultaneously developed in two parallel theses . In this work , the Visual User Interface and the Graphic module for the package has been developed . The Visual User Interface is responsible for all the interaction with the user. It is an interface between the visible actions on the screen and their actual execution for simulation . The software is menu driven and has all the standard features of a Windows package .

ACKNOWLEDGMENTS

I am very thankful to Dr. Anil Mahanta for guiding me in the thesis work . It was mainly because of his support and guidance that I could bring the thesis upto this level .

My wife Priti and my young daughter Ayushi , need a special mention here . They have displayed great amount of patience all along , while I was busy working . Their ever-present support has been instrumental in allowing me to concentrate on the thesis work.

I also wish to express a deep sense of gratitude towards my parents and brother, without whose encouragement and support it would not have been possible for me to come and work here .

Contents

List of Figures

Chapter 1

Introduction

The need for Simulation tools for DSP has been addressed at many levels for quite sometime now .Given the vast gamut of possible applications and the scorching pace of development in DSP and related fields , many of the simulation tools too are being developed to cater for only a specific branch of applications.Some examples could be, non-real time simulations , code-synthesis for real-time hardware, simulation systems for parallel programming ,simulation as training aid and so on.

In the segment of training aids for DSP ,one of the conceptually convenient, but appealing type of simulation system is the block diagram simulation. In fact ,it is possible to develop block diagram simulation tools which , through hierarchical design starting from fundamental levels , are able to represent fairly complex DSP systems . The vastly increased computational performance of desktop computers has made it possible to bring such simulation systems to the common users , who hitherto , had to have access to bigger machines for the desired simulations . Moreover ,the enhanced user- interactiveness of visual platforms like Microsoft Windows serve to add a qualitative improvement to the simulation system .

Block diagram simulation system is perfectly suited to the needs of teaching DSP through experimentation and simulation .More so , since the concepts can be tested using simple simulation schemes involving blocks and viewed through visualization tools which makes the result of a simulation intelligible . Also ,the user is not required to get involved in any kind of lengthy programming effort ,or algorithm development ,instead he can concentrate more on the conceptual aspects and test them from all standpoints .

This provided the motivation to develop a Microsoft Windows based block diagram simulation software ,which would act as a tool for training and provide an

amicable "Design Environment" that manages an application development for conceptualization and experimentation .

Block diagram languages for simulation actually have a long and distinguished history ,testifying to their attractiveness to the DSP community . The DSP concepts can be very neatly represented by blocks in a block diagram system . A number of systems have been built based on this right from 1960's to current times [1].

However , it should be noted that block diagrams are not the only attractive means of high level representations of DSP algorithms . At MIT ,techniques based on functional languages have been evolved[2] . These efforts have supplemented numeric computation with ***symbolic*** computation .In other words , instead of producing ***data*** that represent a signal , the objective is to produce ***expressions*** that represent a signal .Such techniques are very useful for algorithm development and exploration .

1.1 A Brief Survey of DSP Simulation Software

Currently there is a plethora of DSP simulation software ranging from the general purpose ones to those catering for specific applications . Among the block diagram simulation software packages , some of the popular ones are described in the succeeding paragraphs .

- **Hypersignal** series from Hyperception . The simulation software packages from Hyperception (e.g. Hypersignal RT4)cater for a wide range DSP applications like
    - Provision of comprehensive DSP Environment .
    - Digital Filter Design and DSP code generation .
    - Time and frequency Domain Analyses .
    - Real time Spectrum analysis .
    - Image Processing.
    - Virtual Instrumentation .

- **Gabriel / Ptolemy** .Developed at University of California ,Berkeley , Gabriel [3] is a Unix based software system intended to manage complete development of DSP applications . It performs non-real time simulations as well as code-synthesis for real-time hardware . It supports hierarchical design of function blocks ,calling the individual blocks as "Stars" and the block diagrams "Galaxy". A Galaxy can further take on appearance of a star or it can itself contain galaxies . It supports users with various levels of sophistication . At the highest level ,Gabriel applications are described as hierarchical block diagrams .At the lowest level ,the user can either simulate the algorithm locally on a workstation , simulate the target architecture running the generated code ,or download the code into hardware and run it in real time .

- The other block diagram simulation systems for DSP include softwares like BOSS(Block Oriented system simulator) [4] from University of Kansas , DSPlay from Burr -Brown and SPW(Signal Processing WorkSystem) from Comdisco .

- MATLAB has also come out with its Windows version ,which has a dedicated Signal Processing tool box for simulation of DSP systems.

- A survey of commercial simulation packages can be found in [5] .

1.2 Objectives and Scope of the Thesis

To develop a visually programmable object -oriented software simulation framework to function as a

- tool for analysing DSP systems being represented by a set of basic blocks .

- teaching aid for fundamental concepts of DSP .

Towards this objective , the overall task can be split into following modules :--

- Development of Visual Interface which acts the front or visible layer of the software.
- Development of function library to support the software .
- Development of Graphic display module .

These will be followed by integration of individual modules.

This software has been developed in two parallel theses . While the Visual Interface and Graphic modules are the thrust area of this thesis work , the function library is the main theme of the work in [6] .

1.3 A Brief Overview

The Block Diagram application is built around a high-level block diagram user interface and low-level function block library. The high-level block diagram user interface includes the main menu, Block Diagram work area, a data flow module, and a number of graphical user interface functionalities. The high-level interface is an independent stand-alone Windows application . However, in the same manner that a hardware breadboard would require discrete components to be functional, the Block Diagram requires the discrete block library to be useful.

An extensive library of functional blocks for setting up a block diagram has been provided. The functional blocks fall into three main categories:

- Input and Signal Generator blocks : Input blocks are responsible for getting their data either by reading a disk file or by generating them internally.
- Processing blocks : These blocks include DSP function blocks like DFT ,Filter blocks ,etc.
- Output blocks : These blocks are required to store the output data generated in a simulation session for subsequent post-processing .

One way of viewing this Block Diagram software package is to look at it as a software breadboard. It would help in testing designs , as also act as a teaching aid to visually clarify some of the fundamental but abstract concepts of DSP and related fields .

The breadboard offers a flexible environment where the user can easily add and remove components as well as their interconnections . In a parallel sense, the Block Diagram offers the user a flexible workspace to rapidly test and verify the efficacy of a particular design using discrete functional building blocks .

1.4 Organization of the Thesis

The thesis is organized under the following chapters :--

- **Chapter 2** : Implementation Methodology . This chapter describes the approach and phases of development of the software . It also includes the overall program flow, to give a detailed scheme of progression of the simulation through its various stages .
- **Chapter 3** : Object Windows Application framework . This chapter covers the details of the development of visual interface for the software . It elaborates on the class structures used in the object-oriented code developed under OWL 2.0 (a class library with Borland C++ ,version 4.0 and 4.02) for the Windows part of the software .
- **Chapter 4** : The Graphic Module . This chapter describes the working of the Graphic module .
- **Chapter 5** : Suggested Enhancements . This chapter outlines the possible enhancements to the package for improved functioning of the Block Diagram Simulation System.
- **Chapter 6** :Conclusion.
- **Appendix A** : OWL classes used in the application .

- **Appendix B** : User's Guide.

- **Appendix C** : A Guided Tour through the Package .This section is basically an aid for starting off easily with the simulation software and is more applicable for the new users of Windows environment . It works through a simple simulation exercise with the help of an example .

- **Appendix D** : The File structure of the Application . The file structure needed for development of this software under Borland C++ IDE (Integrated Development Environment) is described in this appendix .

Chapter 2

Implementation Methodology

From a program developers' perspective , the development of the present software can be categorized into the following major(and almost independent) sections , to be integrated finally to provide the intended functionality .

- **The Windows based Interface** . It entails development of visually programmable object -oriented Simulation framework . Basically ,it is required to cater for the creation and manipulation of all the visible components in the Simulation session . The initial development of each of these components is devoid of any linkage with "Blocks" or "Events" which they would finally represent . To make the entire Simulation system an "event-driven" one , a high level of object-orientation was needed . To meet these requirements and to suit the programming requirements of Windows based modules , this part of the project has been developed using Object Windows Library (OWL 2.0) available with Borland C++ ,version 4.02 .

- **The Function Library** : Although most of the algorithms for functions needed for development of this software are readily available [7 , 8] , a great deal of effort is required to build a customized and comprehensive library to suit the needs of this particular software package .

- Development of Graphic Analysis Module : To study and analyze the plots of time and frequency domain signals .

2.1 Phases of Program Development

The pattern of development of the software has been as mentioned below :--

Phase 1 : Formulation of system requirements and features to be incorporated in the software .

Phase 2 : Concurrent and independent development of the following :

- The GUI , or the Object Windows based interface .
- The function library .

Phase 3 : Development of Graphic Display module .

Phase 4 : Integration of various components of the software .

Phase 5 : Testing and debugging .

2.2 The Overall Program Flow

In this section , the overall program flow for the software is explained in detail . The explanation is given with the help of flow charts in Figures 2.1 through 2.6 .

As a first step of execution ,the MainWindow object, which supports all the attendant functionality is created and displayed after initialization [Figure 2.1].The MainWindow object is created from a class corresponding to it ,in which all the functionalities have been encapsulated through member functions(all classes and their features ,have been delved in greater detail in Chapter 3).Next , all the visible features which have to be displayed at the start of the Simulation session are created , initialized and displayed . These include the Main Menu , toolbar with the button-gadgets (representing the more frequently used menu items for increased ease of operation for user) , and the active workspace (referred to as Client area). Once this has been accomplished, the program now waits for instructions from user , in the form of selection of a menu item. At this point ,the valid selections either selecting a function block , Help, or Exit .The invalid selections pertain to clicking the menu items representing "actions" like Run ,Connect,

Change Parameters ,Close All etc. . In such a case , the system generates a message to warn the user of the selected event's invalidity at this point .

Selection of a function block from menu sets off a sequence of actions within the system .These can be summarized as follows :--

- Creation of a visible block (actually a minimized child window object) and its associated data structure to hold all its parameters(See Figure 2.2).
- Pasting of its corresponding icon resource on the block (each function included in the main menu has a corresponding icon resource for visual representation of that function) .
- If the function needs initial parameters , then invoke the corresponding Dialog Box to take in the same .(The creation of Dialog Box follows the similar path as a Child Window).At this juncture , the focus of activity shifts to the Dialog Box on the screen ,i.e. until the Dialog Box is fed in with the data it needs or "Cancel" item has been clicked in it , no other activity can occur (See Figure 2.3).

After the Dialog Box is closed ,the selection process of a function block is complete, and the system again waits for next command from the user . This continues till all the blocks are selected .

The next step in the Simulation is relative placement of the blocks(achieved by clicking and dragging the Iconic Function block on screen) and then connecting them through graphical line segments representing data flow between the blocks . Inbuilt checks have been incorporated for testing the validity of various types of connections . The connections are automatically updated in the linked list containing details of the connections (See Figure 2.4).

Now ,the program is ready to "run" the Block Diagram created on the screen. Before running , the system ascertains the simulation sample size .The "run sequence" is decided by the system by working backwards from the first connected block(connections need not necessarily be made in the eventual running order) and reaching Input / Signal Generator blocks(Refer Chapter 2 in [6]). Once the "run sequence" is finalized , the Block

Diagram is "run" in the designated order (Figure 2.5).As a consequence ,it generates and stores the data sequences for each block ,which can now be studied in detail through Display module(Figure 2.6). The Display creates graphical plots for the data of that block at which it is called .The Display module has been described in detail later(Chapter 4) .

After the graphical analysis , the user has the following choices :-

- To run the same sequence with a different value of sample size (by altering the value in the Run Dialog box).
- To change parameters of some of the blocks in the Block Diagram ,and then run it again (by clicking at the block whose parameters are to be altered and then selecting "Change Parameters" - this calls the appropriate Dialog box for altering the parameters) .
- To add some more blocks to the Block Diagram and run again . This can be done by repeating the process of block selection(as explained above)at any time during a simulation session .
- To completely close down the Block Diagram and create a new one (using Menuitem "CloseAll").
- Exit from the system (by selecting File | Exit or through the SystemMenu).

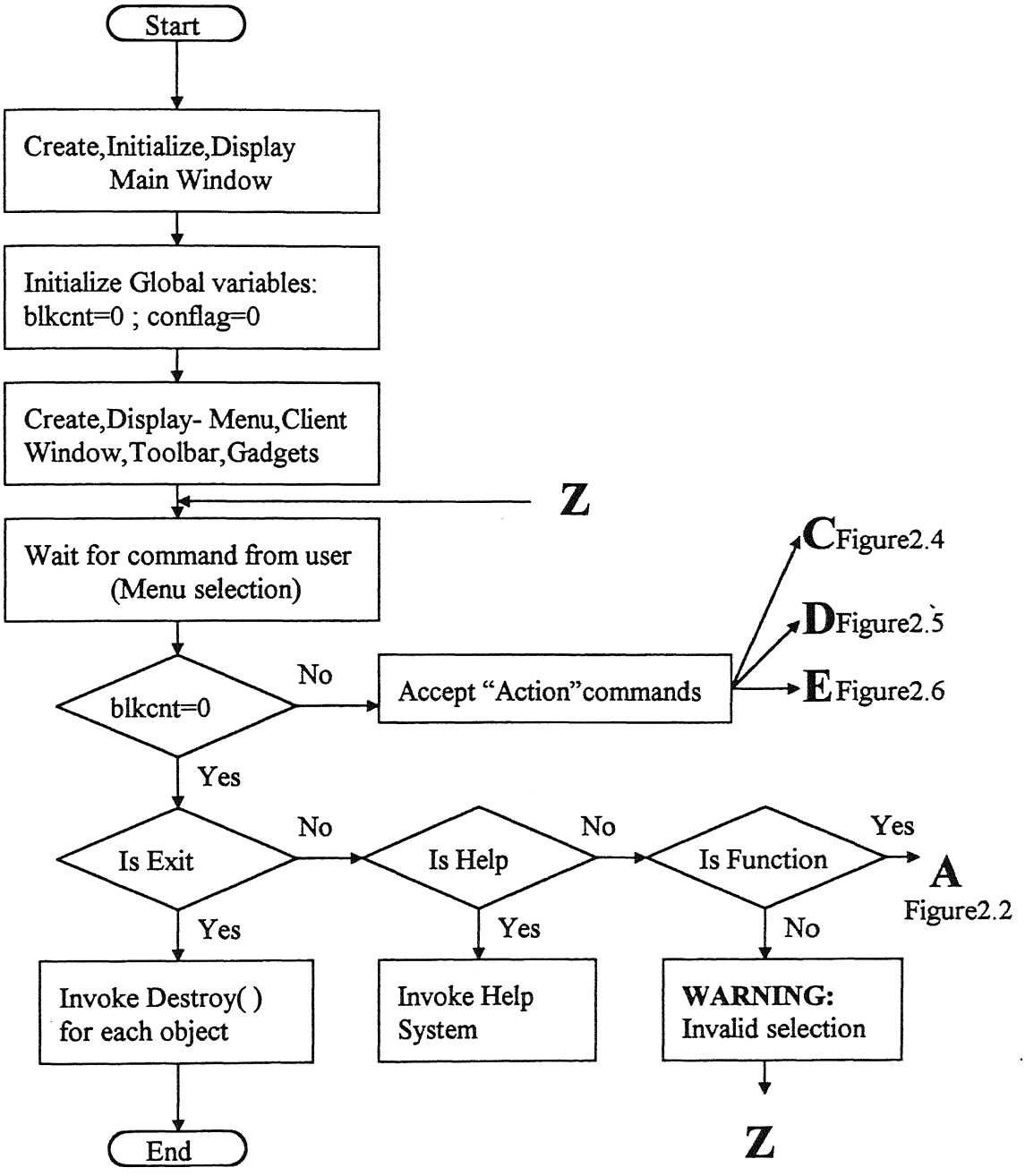


Figure 2.1 : Flow Chart at the Start of Execution of the Block Diagram Simulation Program

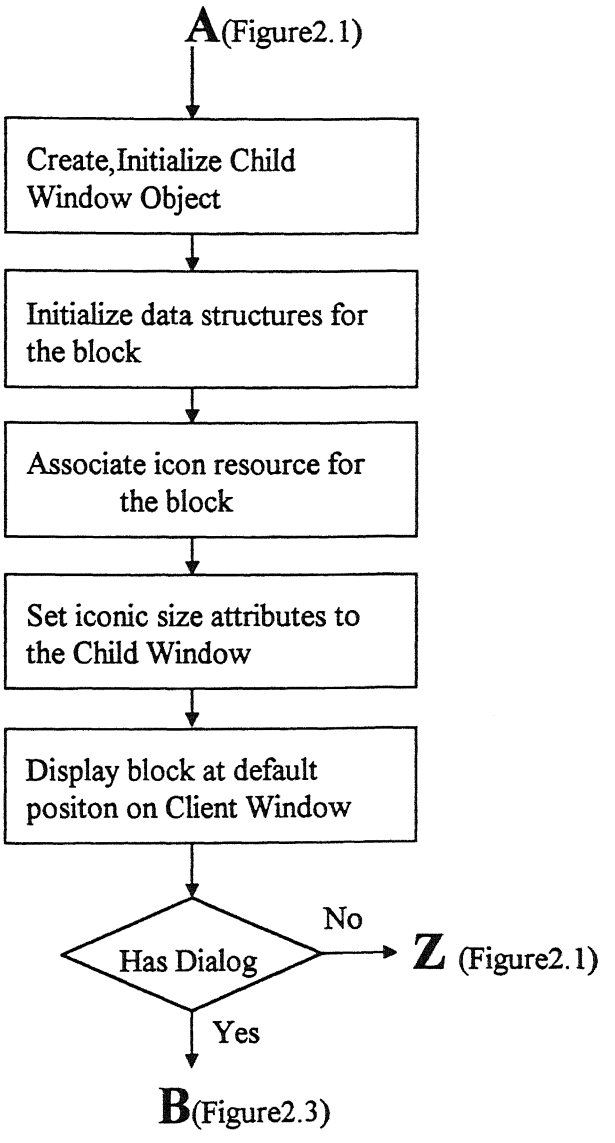


Figure 2.2 : Flow Chart for Block Creation

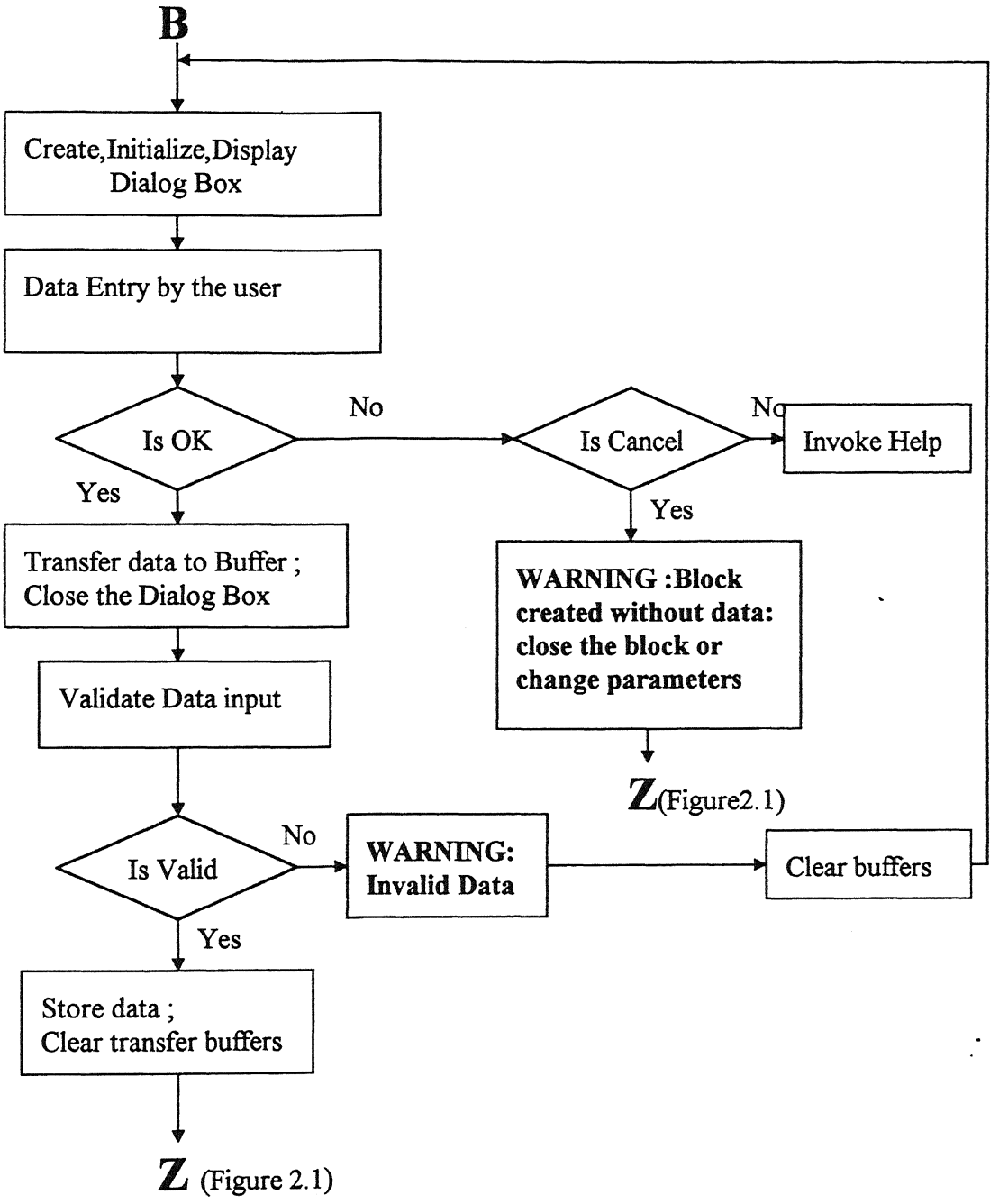
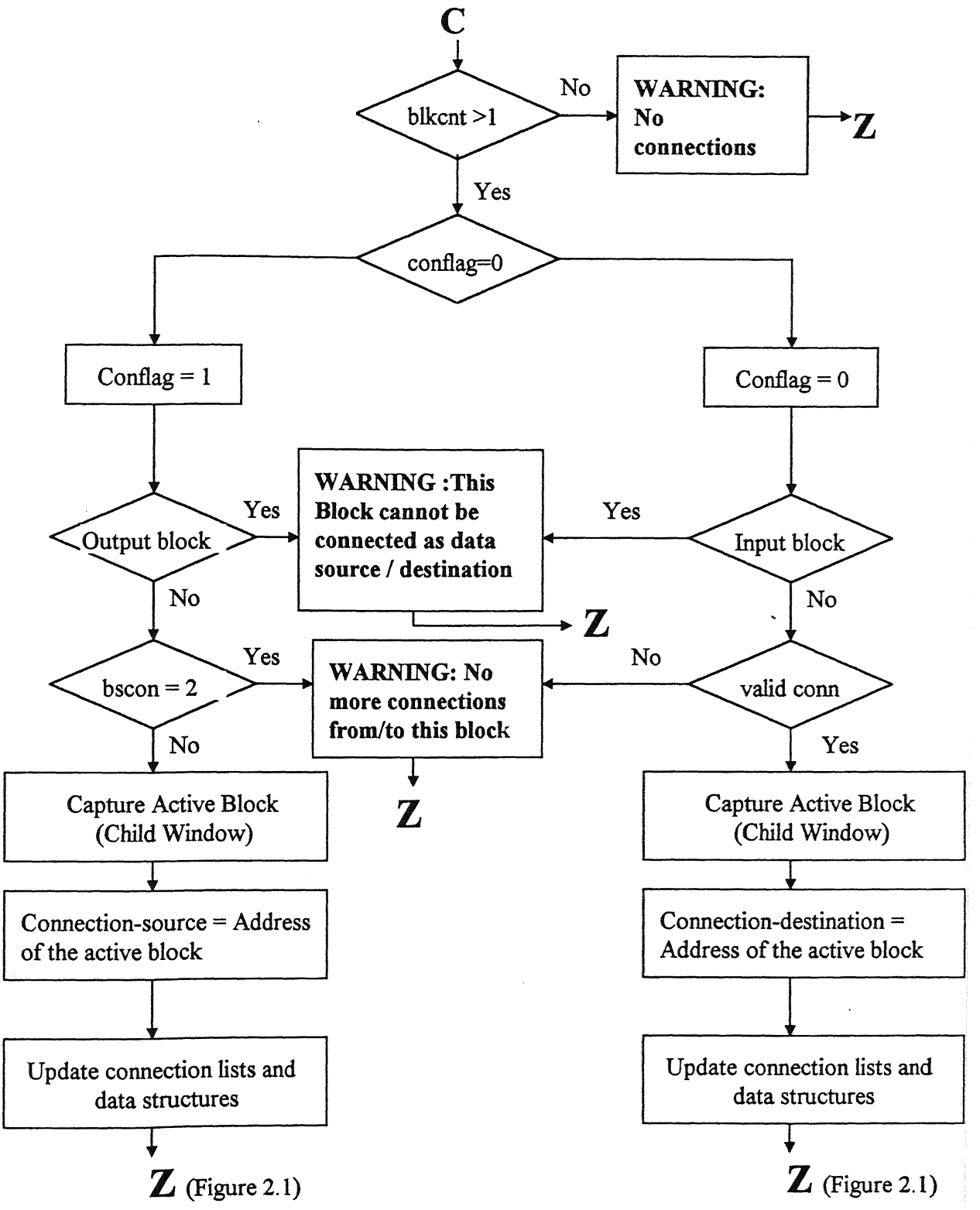
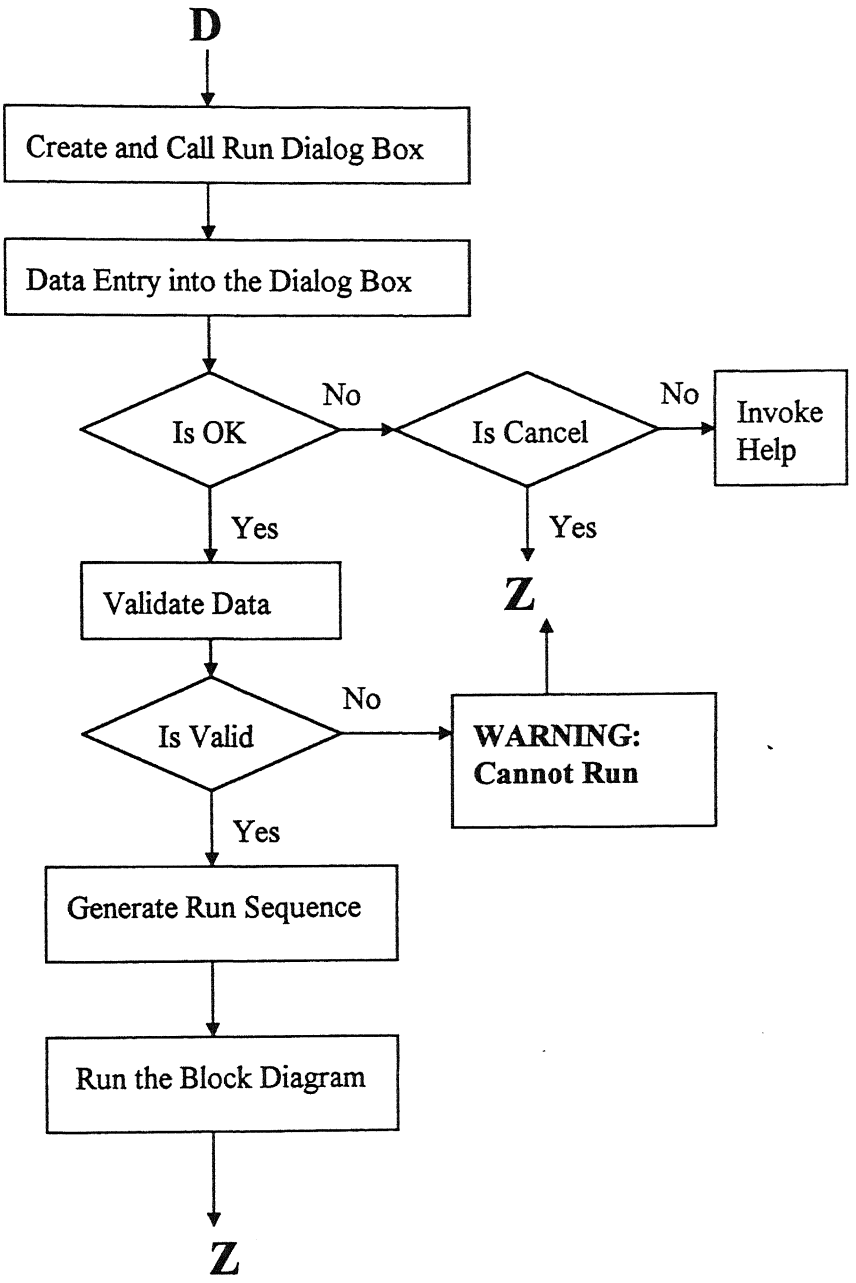


Figure 2.3 : Flow Chart for Creation and Execution of Dialog Box



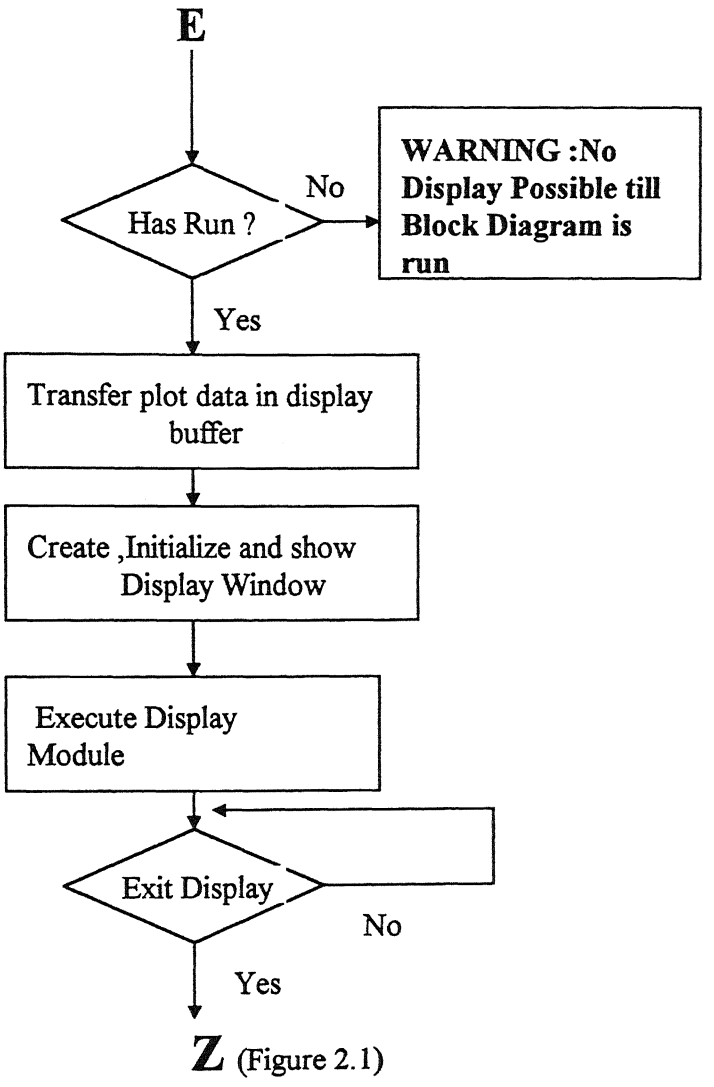
Note : "bscon" checks for blocks connected from source

Figure 2.4 : Flow Chart for "Connect"



Note : For detailed flow chart for sequencing and run algorithms ,refer to Figure in [6]

Figure 2.5 : Flow Chart for "Run"



Note : For details of display module ,refer to Chapter 4

Figure 2.6 : Execution of Display Module

Chapter 3

The Object Windows Application Framework

3.1 Introduction

As seen in the last chapter , the Windows component of this package caters for the Visual User Interface module .This module has been developed almost completely under Object Windows environment .In this chapter ,the development of this module has been discussed in detail. It is to be noted that the name "application" in this chapter , is used to refer to the Visual User Interface module(i.e. the Windows component of the Block Diagram Simulation Software) .

The Object Windows environment provides one of the most robust class libraries for development of Windows applications viz. the OWL 2.0 (Object Windows Library, version 2.0 ,available with Borland C++ ,version 4.0 and 4.02) . OWL superimposes a somewhat higher-level interface onto Windows . Its advantage mostly comes from the application of object -oriented techniques to Windows programming .It would be in order to briefly outline the imperatives of Windows programming environment before the issues involving Object Windows are discussed .

3.2 The Windows Paradigm: The Event -Driven Programming

This type of programming environment is particularly useful where there is a constant need for the environment to react ,in a natural way , to the user actions . If the user clicks a mouse to push a button in a dialog box , task X should be performed . If instead , the user closes a window ,task Y should be performed . If the user switches to a second application , further input should go to the second application while the first lies idle . Another feature of event-driven programming model is that it helps Windows manage several independent applications .If an application asks Windows for the next message , then Windows knows that the application has finished its previous chore .This helps Windows allow some other application to operate . As the focus of user activity

moves from one application to another ,Windows simply routes the user generated messages to the application managing the active window.

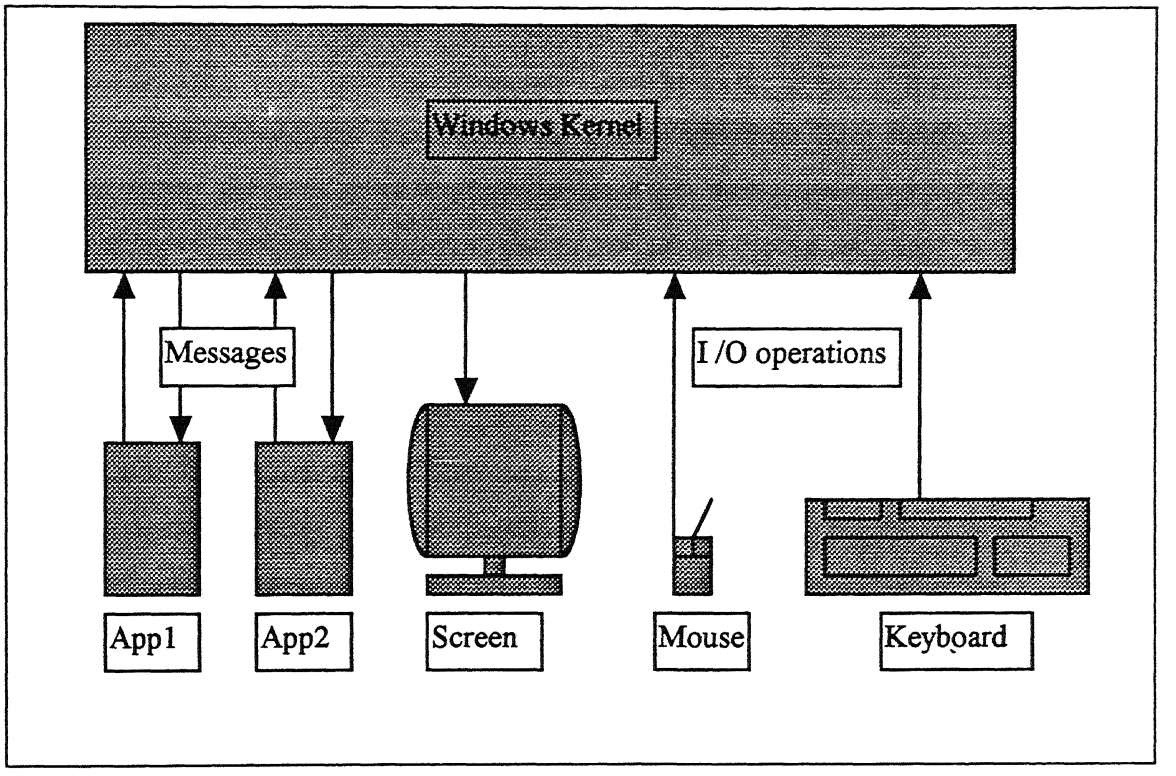


Figure 3.1 : The Event - driven Architecture of Windows

One of the most intricate jobs of Windows environment , and certainly its most visual task , is to manage the display . Because the display is often shared by many applications ,changes made by one application sometime affect another .For example , if the user enlarges the window of one application ,it may partially or fully obscure the window of another .Such tasks are efficiently handled in Windows environment by virtue of its strong "default processing" features and inbuilt memory management .Here, Windows knows that when the obscure window would need to regain the focus , it would send necessary messages for redrawing the window to its original form .

Even though Windows uses an event- based ,message passing architecture ,it also contains a traditional procedural interface . When messages arrive for an application to

process ,much of the response is likely to be a series of procedure calls to the Windows kernel .

3.2.1 Using Resources

Windows is a tremendously rich environment which helps the program developer to manage resources such as windows , dialogs ,menus , controls(buttons ,lists , scrollbars etc), fonts , and so on . Infact , it is the Windows management of these elements(which is otherwise a complex job to accomplish) , that provides an impetus for developing Windows applications. Resources are usually managed in three stages . In the first stage , a program asks Windows to access a resource , e.g. the program might ask to use a certain icon or dialog box or a font (these resources can be both application specific or Windows supplied).If the resource is available ,the Windows gives the program a handle to refer to the resource . In the second stage , the resource is used by the program , usually by passing its handle to a Windows function . Finally ,in the third stage , the program releases the resource , again by using its handle .

3.3 The OWL Paradigm

The Object Windows Library does not fundamentally alter Windows event driven paradigm , but provides a much more troublefree way to use the same. It is an object oriented class library that provides a large number of classes which encapsulate the behavior normally exhibited by Windows applications . The main features of OWL environment can be listed as under :-

- Being a true object -oriented environment ,OWL provides the advantages of abstraction, encapsulation , and polymorphism .
- The Object Windows Library provides classes that relate directly to the major Windows concepts . The customization and addition of behavior is facilitated by ready availability of a number of properties in the base classes , through the inheritance feature of C++ .

- The OWL lets us associate class member functions with individual messages . This eliminates the need for huge , unwieldy C switch statements and conceptually ,it provides a cleaner way to manage messages .

- The code for templatized classes , once created to suit a particular Windows task , are amenable to reuse with very little change for a similar or slightly varying task .

3.4 Development of the Application in OWL Environment

The succeeding paragraphs outline the approach followed for development of the Visual User Interface Module of this software package . It is in line with the general pattern in which OWL based applications are usually developed . There are certain variations, which were necessitated to suit the specific requirements of this particular application and these have been delved into in greater detail at appropriate places .

We can sub-classify the development of this application into three different parts:--

- Formulation of basic class structure ,based on the requirements or the functionalities needed to be incorporated in the application .

- To create the inherited classes from the base classes and customize them by adding specific functionalities to each such class so as to make it perform its designated task .

- Development of resources , such as menus , dialog boxes , icons , bitmaps , and so on .

3.4.1 The Class Hierarchy in the Application

The OWL classes are fairly complete in terms of functionality that could be needed while developing a Windows application . In our application too , a large number of objects have been created using the OWL classes as they are , without any change in them (refer to Appendix A for a brief description of these classes).

Some of the classes , however have been customized to suit the unique requirements of this application .These classes have been inherited from the base classes of OWL and customized by adding / altering the member functions needed to support the application

(Figure 3.2). All such classes have been prefixed with "b3" . The subsequent part of the name of such classes indicates the root class from where this class has been inherited . For example , b3MDIClient class is a customized class derived from TMDIClient class of OWL .

The following description of the customized classes in the application , gives out the major functionalities provided by these classes :--

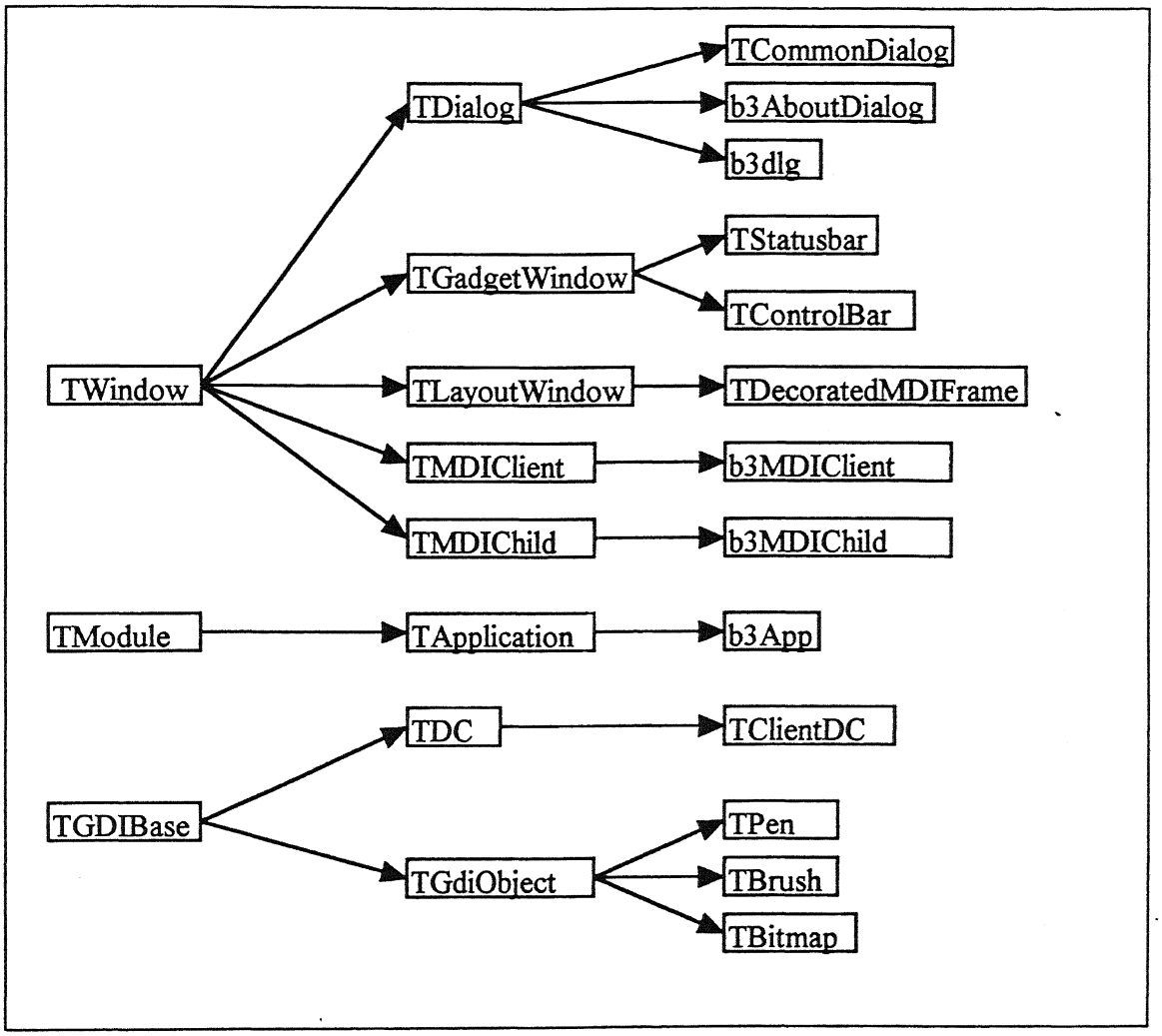


Figure 3.2 : The Class Hierarchy for the Application

- **b3App** : This class has been derived from TApplication class of OWL , and is the cornerstone of the application . It is responsible for creating the main window of the application , and run-time management since it contains the

standard Windows loop that processes the incoming messages . In addition , the following functionalities are performed by the b3App object corresponding to this class :-

- Control and execution of on line Help system .
- Execution of the "Display" module - the module responsible for plotting the data generated in a simulation session . However , b3App only initializes and creates the display Window and then hands over the control to Graphic module, responsible for creating graphs for the data fed to it .
- Display of "About" Dialog Box for the application .
- Closing the application and performing all activities at the time of closing

- **b3MDIClient** : This class has been derived from TMDIClient (itself a derived class from Twindow) .The b3MDIClient object represents MDI (Multiple Document Interface)Client window and it manages all the MDI child windows for the application as their "parent" . Hence , most of the functionality for the application has been put into this class .
  - It has the event handlers for almost all the menu items . Each of these event handlers call the corresponding functions for subsequent execution .
  - The painting logic for the graphic data flow connectivity between two blocks.
  - The simulation of "blocks"which represent a particular mathematical /Signal Processing module ,is done through creation of a unique Child Window in the client area .Thus , the Client Window acts as the Parent Window for each such "block" .
  - It controls the creation of the Dialog boxes of each "block" .

- It has the event handlers for "action oriented menu commands" like "Run" , "Change Parameters" , and "CloseAll" . The functions which are called in response to these events have been made members of this class itself .

- **b3MDIChild** : This class has been derived from TMDIChild and it defines the basic behavior of all MDI child windows used in this application(mostly ,to represent various "blocks" during a simulation session) .For this , the child windows have been tied down to their iconic representation with a specific bitmap pasted on each to represent (visually) a " block" . Like all windows, upon initialization and creation of the child window , all its parameters and required data is stored in a structure created along with it . Whenever ,this block is selected by the user during the simulation session , the pertinent data contained in it is automatically updated .

- **b3Dlg** : This is the class derived from TDialog ,and as the name suggests , it encapsulates all properties displayed by the dialog boxes of the application . It encapsulates the initialization ,creation and execution of all dialog boxes in the application . Each dialog box created using b3dlg class has a corresponding resource definition that describes the placement and appearance of its controls . The identifier of this resource (the dialog resource)is supplied to the constructor of the b3dlg object at the time of its creation .

3.4.2 Implementation of the Classes

Once the classes have been created , their usage is a fairly straightforward mechanism . This has been illustrated in this section with the help of a function which creates a block and then its Dialog Box .

Creation of a Block. The creation process of a block has been explained through an example of say , creation of FIR block . As mentioned earlier , each block on screen is actually a child window ,with a unique identification(Handle) . When the FIR menu item is selected , the response table of the program is searched to find the corresponding function for FIR block creation(these functions have names prefixed with 'Cm' e.g. CmFIR) . The

program execution switches to this function.CmFIR calls the "Create_Child_Block" function which performs the following tasks :--

- It creates a child window object based on the b3MDIChild class , making the "Client Window"(based on b3MDIClient) ,its parent .
- Sets the size attribute to iconic level (32 X 32 pixels).
- Assigns the title of this window as "FIR"(supplied through calling function).
- Assigns the FIR icon resource , as per its identification number .
- Shows the window(the FIR block) at the default position on the screen .

The code segment of "Create_Child_Block" is reproduced below :--

```
void b3MDIClient::Create_Child_Block(LPSTR title , TResId iconResId , int Code)
{
        //Create a new child of type b3MDIChild with given title's attributes.
        b3MDIChild* child = new b3MDIChild(*this, title, 0);
        // Associate ICON w/ this child window.
        child->SetIcon(GetApplication(), iconResId);
        child->Attr.Style |= WS_MINIMIZE ;
        child->Create();
        // Store the details of the child window .
        Bl[ChildCount] =get_block();
        Bl[ChildCount]->CodeNum=Code;
        child->blkptr = Bl[ChildCount] ;
        Bl[ChildCount]->childnum = ChildCount ;
        ChildCount++; .....
```

Hereafter , the CmFIR creates and initializes the data structure to hold all parameters and subsequent data for the FIR simulation , through "new_data_block" function . A detailed description of these can be found in Chapter 2 in[6] .

Creation of Dialog Box . Dialog box creation is initiated through a call to function "calldialog" from CmFIR .

- The dialog box object is created based on the b3dlg class . Its parent window is again the Client Window .
- The Dialog resource for FIR (existing as a resource) is now called through its ID number . The dialog box now physically appears on the screen .
- The b3dlg constructor initializes all edit fields where the user filled data would be copied .
- The user is prompted to fill FIR coefficients in the designated edit boxes through the programmed tab order .
- The parameters filled in by the user have not been validated and are treated just as character strings . They are transferred to a buffer and validated for the permissible range of values . If valid , the parameters are copied into the FIR's BLOCK object(refer Chapter 2 in [6]) , and the FIR block is now ready for simulation .

Chapter 4

The Graphic Display

In this chapter , the Graphic Display Module is described . This module is accessible from any block of the block diagram after it has run .To view the plots of data of a particular block , the user is required to first select that block by clicking on it once and then select the "Display" options from the menu .The popup menu "Display" has 4 commands to select from , each corresponding to the type of plot it provides . These are as under :--

- **Display | Real** . Selecting this menu item would cause Display to plot the real component of the data sequence .
- **Display | Imaginary** . This menu selection plots the imaginary component of the data sequence .
- **Display | Magnitude** . This menu selection generates the magnitude plot of the data sequence . Here , the sample by sample magnitude of data sequence is taken before it is fed to the graphic module . The details of the "magnitude" function are in Appendix A in [6] .
- **Display | Phase** . The data sequence is first used to generate the corresponding phase data which is then plotted .

4.1 The Operating Sequence

The operation of Graphic Display module is explained through an example . The selection of any of the display options from the menu sets off a sequence of actions that occur which culminate at the plotting of the data in desired form . Let us assume that the user wishes to display the "magnitude" plot of the data generated at the end of a block diagram created by him . When he selects Display | Magnitude from the menu , the following actions occur :--

- CmDisplay function is called and its execution starts after verifying that the block whose output is to be plotted , has been run .If not , a message is flashed indicating so and the program aborts CmDisplay and goes back to Wait state for further commands from the user .

- The magnitude of the data sequence is obtained and stored into the buffers for Display module .

- A new popup type of child window is created based on TMDIFrame class of OWL . Popup windows are different from the normal child windows(based on TMDIChild) ,since they are not restricted to the span of their parent window(the MDIClient window) . They can be moved to any part of the screen and their attributes are absolute , unlike the TMDIChild objects whose attributes are relative to the parent attributes . Also , the view of the popup windows cannot be obscured as they always remain on top .However , there is no change in other features .

- The Graphic function is now called to display the plot . After initialization of its data structures , the function finds the maximum and minimum values for X and Y axes .

- The background for the plot is created , i.e. the X and Y axes , grids , and the scales.

- Title of the plot and names of X and Y axes are now written in the Graphic window (these have been passed to this function from the block calling for display) .

- The plot is now drawn on the screen . The X axis range is adjusted to the sample size of the simulation . The plot linearly interpolates between successive points of the Y -data .

- Completion of the plot is indicated by a beep message . Now ,the program waits for the Close command from user , to terminate the plot and close the display window .

- When the user closes the Display window , all the data buffers are cleared and reset and the popup window object is destroyed and the focus is returned back to the main program .

The flow chart on the next page describes the operation of the Display module graphically .

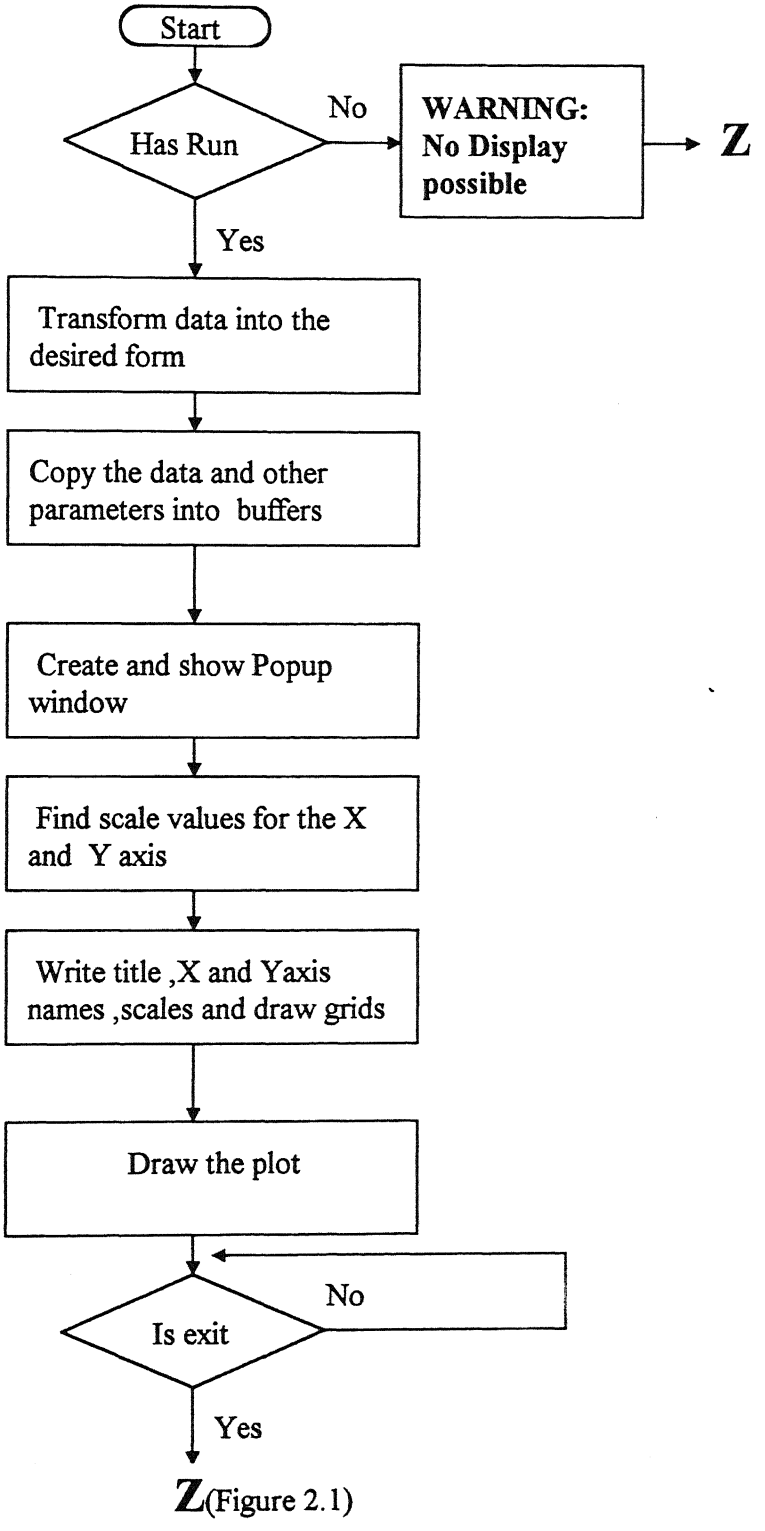


Figure 4.1 : Flow chart for the Display Module

Chapter 5

Suggested Enhancements

At the time of completing this thesis work ,there are a number of features which could not be included in the software ,mostly due to paucity of time .They are covered in the succeeding sections .

5.1 Visual Interface

The most immediate enhancements that would add a new dimension to he functionality of the software are as under :

- **Saving the block diagram in a file** . This feature would facilitate the user to keep a record of the simulations carried out in a session . Essentially ,under the present program structure ,addition of this ability requires amending the b3MDIClient class , b3MDIChild class to make them streamable . The streaming of these two classes would take care of the relative positioning of the blocks on the work window and their interconnections.However ,the associated data /parameters of each block should be handled separately. A module with requisite data structure to hold the complete parametric data pertaining to each block in the block diagram has to be created for that .This can be achieved by creating a generic class with its member functions catering for parameters for all types of blocks .At the time of creation of the object of this class , its constructors initialize all the variables .Those data structures which are modified during streaming out(while saving) ,can indicate this by setting a check flag which becomes the basis for extracting data at the time of streaming in of data (while reading the data from the file ,where block diagram was stored).

- **Incorporating hierarchical design of block diagram .** To implement this feature, it is required to provide the system with ability to save the entire block

diagram as a single block with a unique visual representation . The suggested procedure is outlined here .

To select this new block , it should be called through a file selection procedure , made available to the user through a single new menu command, instead of cluttering up of menu by addition of the names of each such block . Whenever the user wants to select this block ,he should click at this menu item .In response to this menu selection , the "File Open procedure" should be activated and the concerned file of block diagram should be opened . But , for the display of the block diagram as a single new block would require creation of another routine to assign the icon resource , name string in a child window . The rest of the operation of this "block diagram" block is similar to the other blocks except that running sequence would actually be longer due to blocks embedded inside this new block . Thorough testing would be needed to test the memory requirements while running such hierarchical block diagrams . Also needed would be flags for indicating what type of block are these blocks viz. input type(to which no input connection can be made) ,or processing blocks ,or output blocks .

- **Addition of a new block by the user .** Incoroporating this feature is likely to be a more involved procedure . Under the present organization of the program, the various components which go into creation of a block with all its data structures are compartmentalized into different classes depending on their functionalities . The various tasks performed by these blocks during a simulation session have been distributed over different files and routines . Automatic updation of these would need a whole new algorithm to be developed .In addition are the amendments and additions involved in the resources of the system i.e. , updation of menus ,and the consequent identifier scheme (which binds the resources with the main program) .

However , for users with access to the source code of this program , Borland C++ version 4.0 or 4.02 and basic knowledge of Windows

programming, following simple steps can help add new blocks to the function library , taking cue from the way existing blocks have been created :--

- Create the icon resource for the new block using the Resource Workshop .
- Amend the main menu to add the name of the block in the correct place in the menu .
- Create a new dialog box using the Resource Workshop for the parameters of the block .
- Enter a new compiler directive in the file Codenum.h for allotting the block a new codenum . Care should be taken to assign the new codenum an integer value which belongs to the same range as blocks of the same type(Input blocks have a different range for codenums as compared to those of processing blocks and so on).
- Amend the response table of the menu(in b3mdiclient.cpp and its header file).
- Add CmXXXXXX function corresponding to the new menu item , in the b3mdiclient.cpp file , on same lines as the existing CmXXXXX functions. Inside this function ,add the following :--
  - Implement the commands for calling routines for creation and initialization of child block , and association of the icon resource with it .
  - Call the function "new_data_block" to create the data structure to hold its data during simulation .
  - Call the associated dialog box routine giving the identifier of the resource .
  - Amend the "C" language code segment of the function to make it similar in structure to the other existing functions (in b3system.cpp file) and add it in the same file.

- Amend the case-switch statement to cater for this additional function in the function "Run_Functions" in b3MDIClient.cpp file .

The process of addition of a new block is now complete . Now , recompile the entire project in Borland IDE to get the modified .exe file .

5.2 The Display Module

A number of features can be added to the Graphic Display module to increase its power . Some of these are mentioned below :--

- **Simultaneous display of two graphs** .This feature would the Display a better visualization tool . Instances for this kind of display are needed to plot real and imaginary components of a data sequence or magnitude and phase plots of a sequence .

- **Zooming** . This feature would make it possible to study a part of the plot in greater detail .

- **Printing the plot** .

Chapter 6

Conclusion

In this thesis , Object Windows based Visual User Interface for the Block Diagram Simulation software has been developed . With the usage of Windows platform and addition of many customized features , it has been attempted to make the software as user friendly as possible . The Visual Interface has been developed under OWL 2.0 ,thus making it quite robust . Being C++ code ,it has inherent modularity and adding new features would only invlove addition of the code in required files without alterating the present structure .

It is felt that the software would meet its intended objective of being an effective training aid for fundamental concepts of DSP .However , additional features like hierarchical design for simulation , expansion of function library (to include image processing and 2D functions) and improvements in Dislay module would enhance the power of this package substantially .

Bibliography

[1] Gold B ,and Rader C , "Digital Signal Processing of Signals" *McGraw Hill ,New York* , 1969

[2] -------, "The Signal Representation Language" *,IEEE Transactions ASSP , vol. -33 ,* Aug 1985

[3] Lee E.A , Wai-hung Ho ,Goei E.E , Bier J.C , Bhattacharya S , "Gabriel : A Design Environment for DSP" , *IEEE Transactions on Acoustics ,Speech , and Signal Processing , Vol. 37 , No. 11 ,* November 1989.

[4] Shanmugan K.S , "Simulation and Implementation Tools for Signal Processing and Communication Systems" , *IEEE Communications Magazine* , July 1994 .

[5] Kornbluh K , " Active Data Analysis : advanced software for the'90s", *IEEE Spectrum* ,November 1994 .

[6] Warrier S , "Windows based Block Diagram Simulation for DSP Systems: Implementation of Simulation Functions" , *M Tech Thesis ,IIT Kanpur* ,March 1995 .

[7] "Reference Guide , Borland Object Windows for C++ ,version 2.0" , *Borland International ,Inc.*,1993 .

[8] "Programmer's Guide , Borland Object Windows for C++ ,version 2.0" , *Borland International ,Inc.*, 1993 .

[9] Embree Paul , Kimble Bruce , " C Languages Algorithms for Digital Signal Processing", *Prentice Hall , Inc.* , 1991

[10] Press WH, Flannery BP, Teulolsky SA, Vetterling WT, "Numerical Recipes in C : The Art of Scientific Computing", *Cambridge University Press* , !988.

Appendix A

OWL Classes used in the Development of Visual User Interface

The list of all OWL classes used in the development of the application are presented here in an alphabetical order. Alongwith a brief description of these classes ,the header file containing that class has been specified .

TApplication Class

Header File applicat.h

Derived from TModule, TApplication acts as an object-oriented stand-in for a Windows application module. TApplication and TModule supply the basic behavior required of a Windows application. TApplication member functions create instances of a class, create main windows, and process messages.

TBrush Class

Header File gdiobjec.h

The GDI Brush class is derived from TGdiObject. TBrush provides constructors for creating solid, styled, or patterned brushes from explicit information.

TButtonGadget Class

Header File buttonga.h

Derived from TGadget, TButtonGadgets represent buttons that can be clicked on or off. TButtonGadget objects respond to mouse events in the following manner: when a mouse button is pressed, the button is pressed; when the mouse button is released, the button is released. Commands can be entered only when the mouse button is in the "up" state. When the mouse is pressed, TButtonGadget objects capture the mouse and reserve all mouse messages for the current window. When the mouse button is up, button gadgets release the capture for the current window.

TCommonDialog Class

Header File commdial.h

Derived from TDialog, TCommonDialog is the abstract base class for TCommonDialog objects. It provides the basic functionality for creating dialog boxes using the common dialog DLL.

TDC Class

Header File dc.h

TDC is the root class for GDI DC wrappers. Each TDC object has a Handle protected data member of type HDC (handle to DC). Win API functions that take an HDC argument can therefore be called by a corresponding TDC member function without this explicit handle argument.

TDecoratedMDIFrame Class

Header File decmdifr.h

Derived from both TMDIFrame and TDecoratedFrame, TDecoratedMDIFrame is an MDI frame that supports decorated child windows. TDecorated MDIFrame supports custom toolbars.

TDialog Class

Header File dialog.h

TDialog objects represent both modal and modeless dialog box interface elements. (A modal dialog box disables operations in its parent window while it is open.) A TDialog object has a corresponding resource definition that describes the placement and appearance of its controls. The identifier of this resource definition is supplied to the constructor of the TDialog object. A TDialog object is associated with a modal or modeless interface element by calling its Execute or Create member function, respectively.

TEdit Class

Header File edit.h

A TEdit is an interface object that represents an edit control interface element in Windows. A TEdit object must be used to create an edit control in a parent TWindow. A TEdit can be used to facilitate communication between the application and the edit controls of a TDialog.

TFrameWindow Class

Header File framewin.h

Derived from TWindow, TFrameWindow controls such window-specific behavior as keyboard navigation and command processing for client windows. For example, when a window is reactivated, TFrameWindow is responsible for restoring a window's input focus and for adding menu bar and icon support. TFrameWindow is a streamable class. In terms of window areas, the frame area consists of the border, system menus, toolbars and status bars whereas the client area excludes these areas. Although frame windows can support a client window, the frame window remains separate from the client window so that you can change the client window without affecting the frame window. ObjectWindows uses this frame and client structure for both TFrameWindow and TMDIChild classes. Both of these classes can hold a client class. Having a separate class for the client area of the window adds more flexibility to the program.

TMDIChild Class

Header File mdichild.h

TMDIChild defines the basic behavior of all MDI child windows. Child windows can be created inside the client area of a program's main window. Because child windows exist within the parent windows' pixels, they must have the parent windows previously defined. For example, a dialog box is a window that contains child windows, often referred to as dialog box controls. To be used as MDI children, classes must be derived from TMDIChild.MDI children can inherit keyboard navigation, focus handling, and icon support from TFrameWindow. TMDIChild is a streamable class.

TMDIClient Class

Header File mdi.h

Multiple Document Interface (MDI) client windows (represented by a TMDIClient object) manage the MDI child windows of a TMDIFrame parent.

TPen Class

Header File gdiobjec.h

TPen is derived from TGdiObject. It encapsulates the GDI pen tool. Pens can be constructed from explicit information or indirectly.

TWindow class

Header File window.h

TWindow, derived from TEventHandler and TStreamableBase, provides window-specific behavior and encapsulates many functions that control window behavior and specify window creation and registration attributes.TWindow is a generic window that can be resized and moved. An instance of Twindow can also be constructed, though normally TWindow is used as the base class for specialized window classes.

TMDIChild defines the basic behavior of all MDI child windows. Child windows can be created inside the client area of a program's main window. Because child windows exist within the parent windows' pixels, they must have the parent windows previously defined. For example, a dialog box is a window that contains child windows, often referred to as dialog box controls. To be used as MDI children, classes must be derived from TMDIChild.MDI children can inherit keyboard navigation, focus handling, and icon support from TFrameWindow. TMDIChild is a streamable class.

TMDIClient Class

Header File mdi.h

Multiple Document Interface (MDI) client windows (represented by a TMDIClient object) manage the MDI child windows of a TMDIFrame parent.

TPen Class

Header File gdiobjec.h

TPen is derived from TGdiObject. It encapsulates the GDI pen tool. Pens can be constructed from explicit information or indirectly.

TWindow class

Header File window.h

TWindow, derived from TEventHandler and TStreamableBase, provides window-specific behavior and encapsulates many functions that control window behavior and specify window creation and registration attributes.TWindow is a generic window that can be resized and moved. An instance of Twindow can also be constructed, though normally TWindow is used as the base class for specialized window classes.

Appendix B

Users' Guide

Most of the contents of this User's Guide are also available in the "Help" for the Block Diagram Simulation ,which you can access by clicking at the **Help | Contents** menu item in the Main Menu.

The contents of Users' Guide are arranged in the following manner :

B.1 Starting the Application

B.2 Menu Commands .

- File
- Functions
- Connect
- Change Parameters
- Run
- Display
- CloseAll
- Help
- Using the Block Diagram Menu Ribbon Commands .

B3 Using the Block Diagram Simulation Software .

- Selecting and Positioning the Blocks in a Block Diagram Worksheet
- Connecting the blocks for a block diagram
- Modifying the Parameters of a Block

- File input for data or Filter parameters
- Running a Block Diagram
- Starting a new Simulation Session

B.1 Starting the Application

DSP Block Diagram Simulation can be started in Windows environment in any one of the following ways :

- Click on Program Manager's **File | Run** menu item . A dialog box would appear on the screen asking for the name of the .exe file to run . Key in the file name "b3.exe" with its path .e.g. "d:\dsp\b3.exe"(if b3.exe is placed in dsp directory of D drive).
- Click on the icon of "DSP Simulation" placed in the group "WorkGroup" in Program Manager of Windows .

B.2 Menu Commands

Following is the complete list of the menu commands implemented in the system .To select any menu tiem ,you can either use mouse to click(left button) on the appropriate menu item or use Alt + Character(Character implies the alphabet in the Popup menu name which is underscored) to open the popup menu and then navigate up and down using arrow keys till you reach the desired menu item .Then, press Enter to select that select any menu command.

- **File .** The popup menu "File" contains only the "save" and "Close" command, since other operations like File |open and File | new have not been implemented in the software as yet .
  - **Save .** Select this command for saving a data sequence generated by the block diagram or by any individual block in a simulation session.
  - **Close .** This menu item causes the application to close . Choose this item to exit from the DSP Block Diagram Simulation .

- **Functions .** This popup menu contains all the functions presently available in the function library . Those menu items having further sub-menus have a small arrowhead on the right corner .

  The following is the list of all the functions ,presently available in the system. For a detailed description of these functions ,refer to the Appendix A in[6].

  - Signal Generators
    - Sinusoid
    - Damped Sinusoid
    - Step
    - Impulse
    - Pulse
    - Ramp
    - Gaussian Random Sequence
    - Uniform Random Sequence
    - Poisson Random Sequence
  - Arithmetic blocks
    - Adder
    - Subtractor
    - Multiplier
    - Divider
    - Gain
    - Power
    - Conjugate

- Absolute Value
- DSP Functions
  - FIR Filter
  - IIR Filter
    - Direct Form
    - Cascade
    - Parallel
    - Pole Zero
- DFT
  - FFT
  - IDFT
  - IFFT
  - Convloution
    - Time Domain
    - Frequency Domain
  - Correlation
  - Delay
  - Interpolation
  - Decimation
- Transcendental Functions
  - Sine
  - Cosine
  - Tan

- Inverse Sine
- Inverse Cosine
- Inverse Tan
- Sine Hyperbolic
- Cosine Hyperbolic
- Tan Hyperbolic
- Inverse Hyperbolic Sine
- Inverse Hyperbolic Cosine
- Inverse Hyperbolic Tan
- Exponential
- log

- Windowing Functions
  - Rectangular
  - Bartlett
  - Blackman
  - Hamming
  - Hanning
  - Kaiser

- Statistical Data
  - Second Order Statistics
  - Nth Order Statistics

- **Connect** : This menu Command comes into play after you have placed all selected blocks at the appropriate place on the screen . Select this command to connect any two blocks .

- **Change Parameters** : This command is used to modify the parameters of any block of the block diagram . Click this command after clicking on the block whose parameters you wish to modify .

- **Run** : After the connections are completed to create a block diagram ,select this menu item to start the run sequence .

- **Display** : This command is for invoking the Graphical Display Window which will show the plot of data at the point in block diagram from where Display was selected .The display can be viewed, one at a time for any of the following components of the data generated by a block during Run .

  - Real
  - Imaginary
  - Magnitude
  - Phase

- **Close All** : This command will clear the present block diagram on the screen and after that ,the system is ready to start another Simulation session by creating a new block Diagram .

- **Help** : This command invokes the Help for block diagram Simulation .

Using the Block Diagram Menu Ribbon Commands . In addition to the Block Diagram menu commands, a menu ribbon with selectable button commands is available.These iconic Buttongadgets represent the more frequently used menu items during a Simulation Session. The menu ribbon commands will perform exactly as their menu command counterparts, but provide an easier method

of selection. To select a menu ribbon command , place the mouse cursor over the desired command button and choose it with a single-click of the left button of the mouse.

Note : A brief description of each menu command appears on the MessageBar in the bottom of the Main Window as you select / highlight a menu item ,or when Mouse points to one of the buttons on the Menu Ribbon .

B.3 Using the Block Diagram Simulation Software

This section describes the method for setting up a block diagram and then running it.

- **Selecting and Positioning the Blocks in a Block Diagram Work Window .**

  To select a block ,just select the concerned menu item from the Function popup menu , or from the menu ribbon. A block representing the selected function would appear on the bottom left of the Working Window .This software allows for a very free-form method of positioning the selected blocks. After selecting the blocks ,simply click and drag (using the mouse) each block to the desired position on the screen for subsequent processing.

  After the blocks are all in position, you are ready to connect the data flow of each block.

  Note :You must not move a block AFTER it has been connected.

- **Connecting the blocks .**

  After positioning the blocks on the Working Window at the desired places ,you can start connecting the blocks . To connect any two blocks ,first click with your mouse at the source block (i.e. in the direction of data flow - from source to destination) ,then press "Connect" in the menu ,followed by the "Destination" block and then press "Connect" again .The point to note is that

,while connecting any two blocks ,one outputs the data to the other .So ,first select the block whose output would be carried to the next block through the connection made by you. The overall "clicking" sequence for connecting any two blocks is, therefore , "Source Block - Connect - Destination Block - Connect" .

- **Modifying the Parameters of a Block**

  Modifying parameters of the individual blocks which make-up a block diagram algorithm can be accomplished by selecting the Menu Command "Change Parameters" ,any time during a Simulation Session. If the particular block has user-controllable parameters, a dialog box will open to allow you to modify specific block parameters. After you have modified the parameters and chosen the OK button within the block's dialog box, you may run the Block Diagram simulation by using the "Run" Menu command.

- **File input for data or Filter parameters**

  If you wish to provide input data to the Block Diagram through File , select the **Functions |Signal Generators |File Input** menu command . This will create a similar block called "File" on the Working Window and then open a Dialog Box for you to enter the name and path of the File for the data . This file must contain data in a pre-defined format [6]for the Block Diagram to utilize it, else a Message will be flashed on the screen regarding it being Invalid Data .

  For Filters, if you wish to input the coefficients through a File , select the File Button in the Dialog Box for Parameters of the Filter ,which appears on screen immediately after you select a Filter menu item .A dialog box would appear on the screen for you to enter the filename and the path for fetching the filter order. This file ,too, should contain the data in a pre-defined format for it to be used by the Block Diagram [6] .

- **Running a Block Diagram**

Once ,you have connected the Blocks in a Block Diagram ,you can run it by pressing "Run" Menu Command . On selecting "Run" ,a Dialog box would appear asking you to specify the number of samples ,you wish to run it for .Press OK after entering the value in the Dialog box to start running the Block Diagram . The Run sequence is generated internally by the system by starting from the final block and going backwards till it reaches one or more Signal Generator blocks(which do not take any input ,but just generate a data sequence) .Then, it starts running from the Input blocks till the end . A Message Box at the end of Run sequence informs you of the message "Run blocks Complete" .

- **Viewing Graphical Display** .

After you complete the Run of a Block Diagram , you can call the Display to view graphically ,data sequence at the end of any block . Just click on the block at which you wish to view the Graphical plot ,and then select Display |<options> (options imply one of the four components of the data for which plot would be displayed, viz. Real ,imaginary ,Magnitude ,Phase) from the menu . The Display Window appear on the screen and it would show the plot of the data sequence as per the selection .

Note : Close the current Display Window before you view the next plot .

- **Starting a new Simulation Session**

Once you are done with a particular block Diagram Simulation Session , you can erase the block diagram on the screen and start a fresh block diagram for simulation by selecting the **Close All** command from the menu .

- **Exiting from Block Diagram Simulation**

To exit from a Simulation Session and to close down the Block Diagram Simulation , select **File | Close** from the menu or select "**Close**" from the Main Window's System Menu (To activate any system menu , press Alt-Spacebar or left click on the top left corner window with a bar) .

Appendix C

A Guided Tour through the Package

This section is meant for users who are not familiar with Windows environment . It is recommended to read this section in conjunction with the Users' Guide(Appendix B). The method for using the software has been illustrated through a simple example of block diagram simulation . The description has been supported by plates depicting images of the active screen when the software is run .

When the software is started (either using File | Run or by clicking the DSP Simulation icon) the screen for Simulation appears in the maximized form . The screen size can be adjusted by the user to suit his requirements . The main window of the application displays the menu and menu ribbon on top , and the message bar in the bottom . The white screen area is the Working Window (Client Window) for simulation . The functions can be selected by the user from the menu (Figure C1) . They can be selected from the menu ribbon also ,if their icons have been provided . To select a function from the menu ribbon ,just click on the appropriate icon with the left mouse button .

As a consequence of selection of a function from the menu , the block representing that function appears in the left bottom of the Working Window . If the function needs initial parameters to be given by the user , a Dialog Box appears on the screen . The user can fill in the parameters in the edit boxes of the Dialog box (Figure C2) .Forward movement from one edit box to another is possible by pressing 'TAB' on the keyboard and backward movement by 'SHIFT+TAB' . Once ,the parameters have been filled in , the user should press 'OK' button in the dialog box . If the data is not valid , the system will flash a message accordingly and the dialog box will reappear . If the user does not wish to use the selected block , he can press cancel and then close the block also (by selecting "Close" from the system menu of the block) .

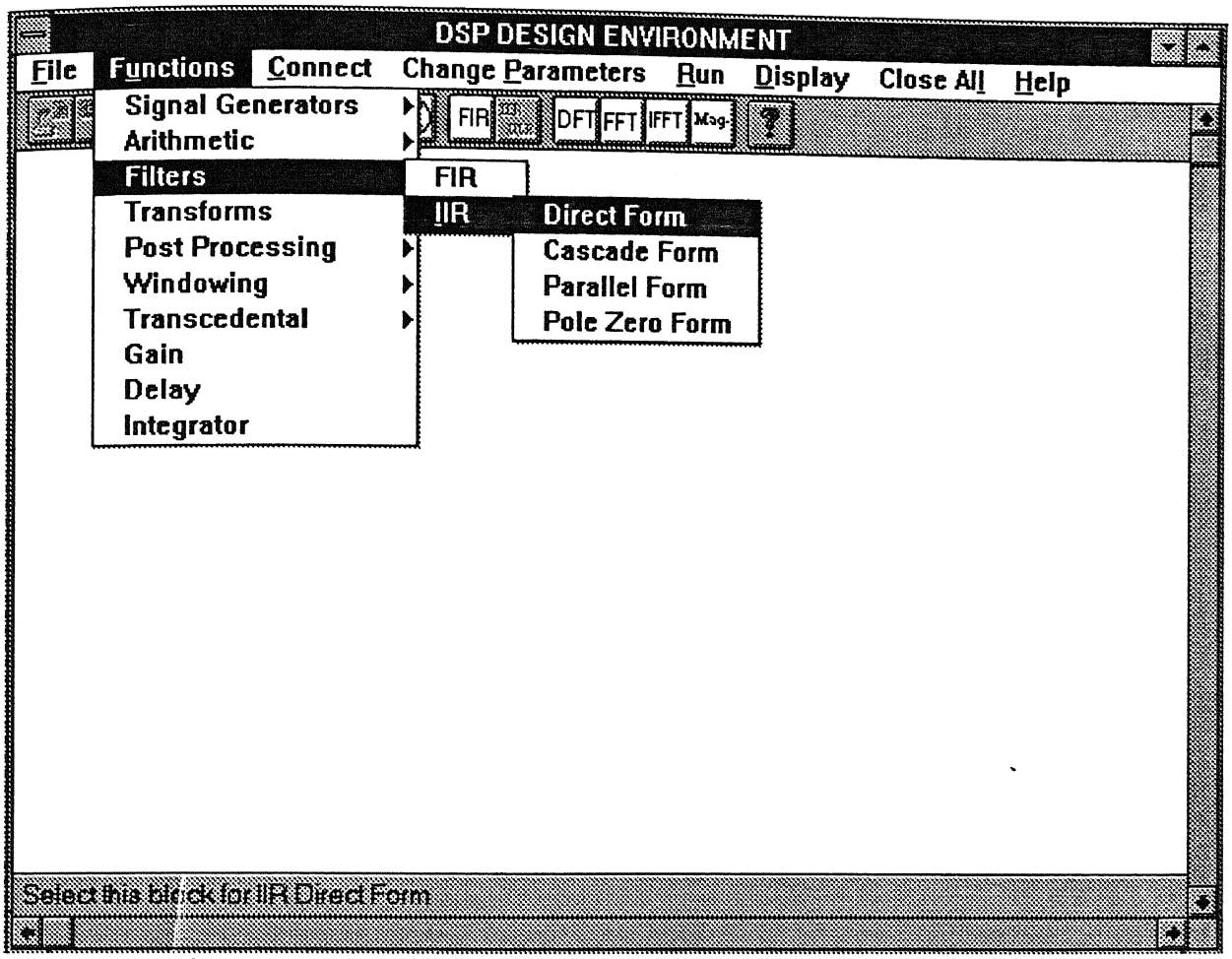


Figure C1 : Selection of blocks for Simulation through Main Menu of the Application

This procedure continues till all the desired blocks have been selected . Then the user can move the blocks to any position on the Working Window , and connect them (refer Users' Guide for procedure for connecting two blocks) to create a block diagram .

An example of a simple block diagram can be seen in Figure C3 . Here , two sinusoids(added with Gaussian noise) have been added together and then autocorrelated . FFT of the autocorrelated waveform is taken and its magnitude is viewed through the Display(Figure C4) .

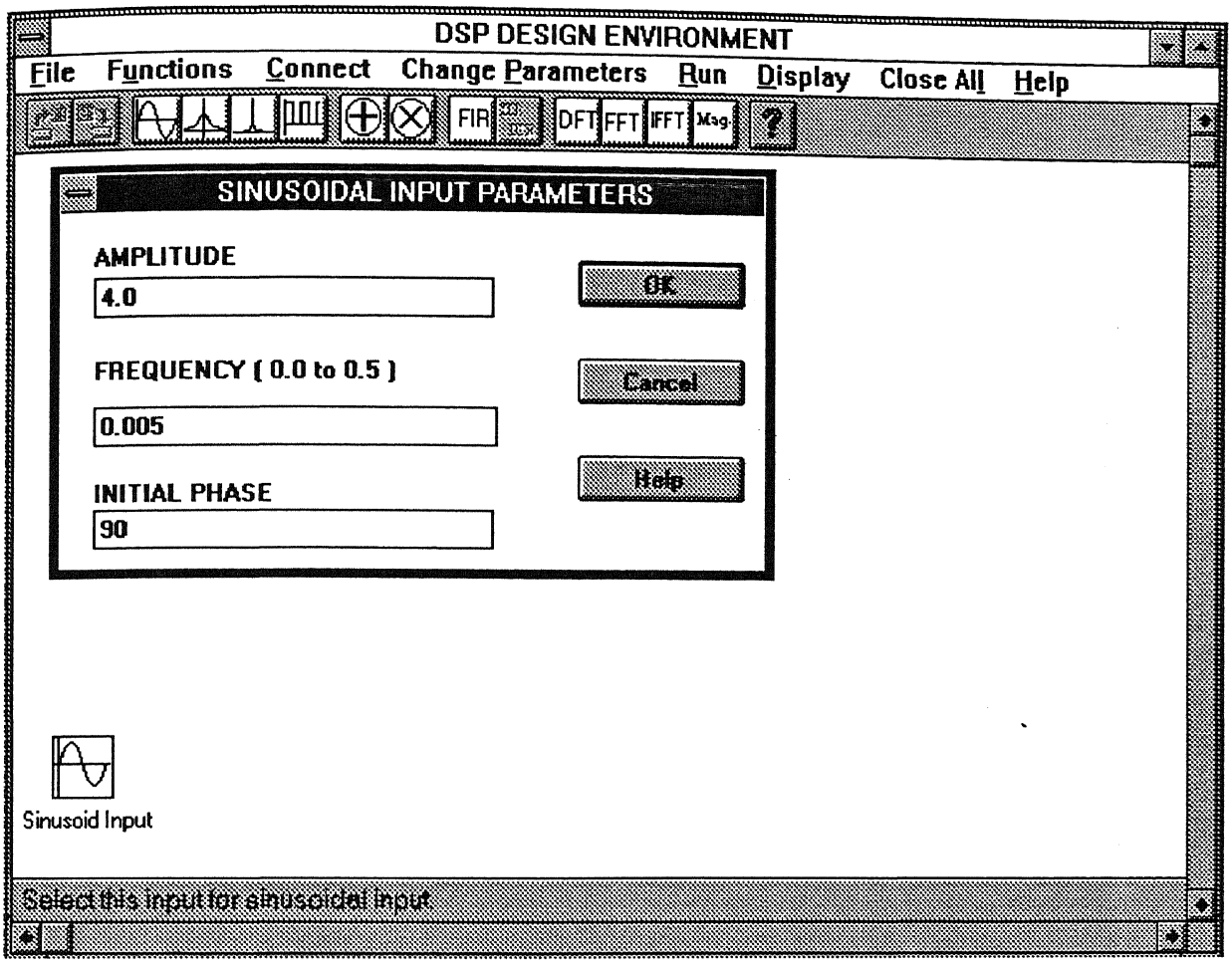


Figure C2 : Once a block is selected ,its initial parameters can be filled in by the user through its Dialog Box .

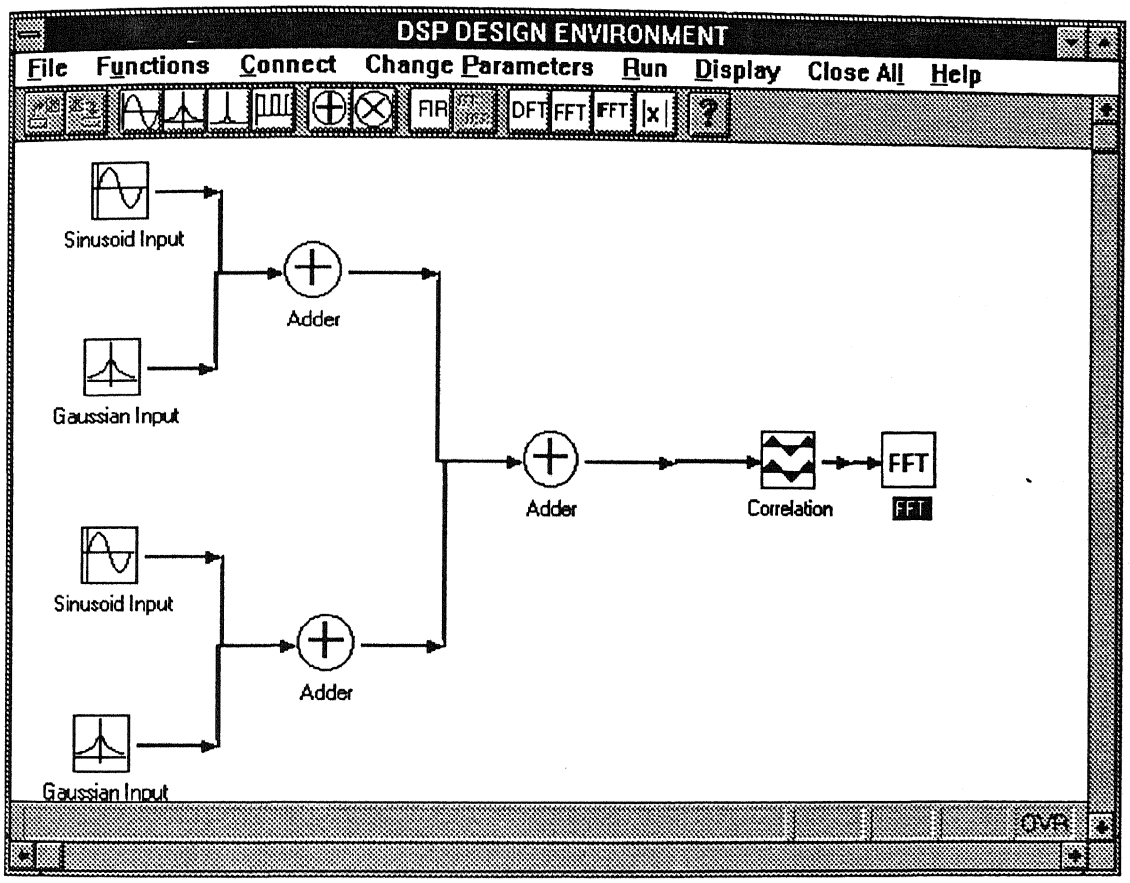


Figure C3 : The Block Diagram

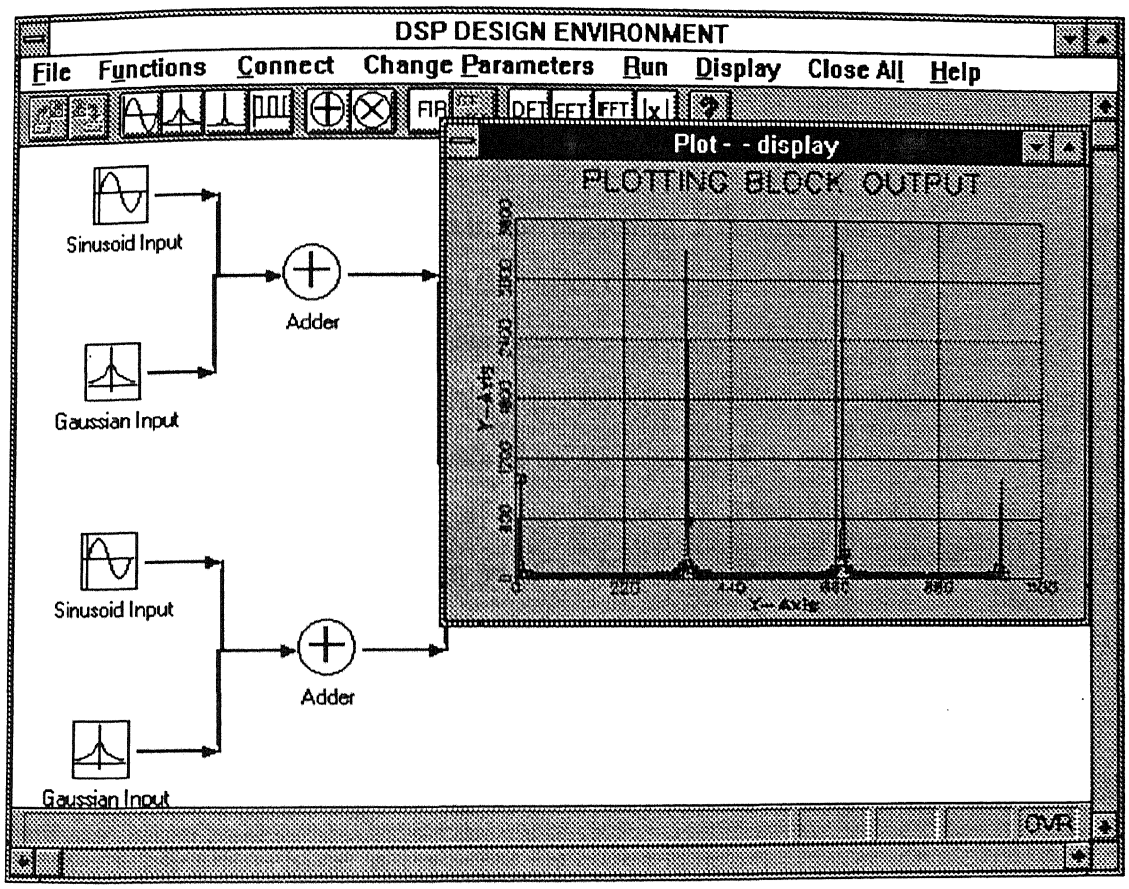


Figure C4 : View of Display of the Magnitude of the data at the output of the block Diagram

Appendix D

The File Structure

The entire source code has been spread over files of different types . Their details are given in the succeeding paragraphs .

The .cpp files and .h files : The .cpp (C plus plus) files contain the code for implementation of classes (defined in corresponding .h files) used in the application . The names of each of these files indicate the major class implemented in it ,e.g . b3mdicln.cpp has all the code for b3MDIClient class and its member functions .

Resource data files : The resources for the application have been created using Borland Resource Workshop and stored in .ico files (for icon resources), .bmp files (for bitmap resources) and .rc file (for all other resources) . These files are compiled separately by Borland Resource Compiler into .res file which is linked to the .exe file finally .

Help file (.hlp) : This file is obtained after the .rtf files (Rich text files) are compiled by Help Compiler HC31 using the help project file (.hpj) .

Module definition file (.def) : The default .def file for Windows has been used . This file contains the settings for standard memory options used by the application ,like the STACKSIZE ,HEAPSIZE and whether the code and data are of type PRELOAD and MOVEABLE or not .

Project file (.ide) : This file contains all the compiler options ,linker options and other system options used in the project . The specific options needed by each file forming part of the project is stored in this file . While in IDE , loading this file first (while opening the project) , would set up all the necessary options needed for compilation process for the entire project .

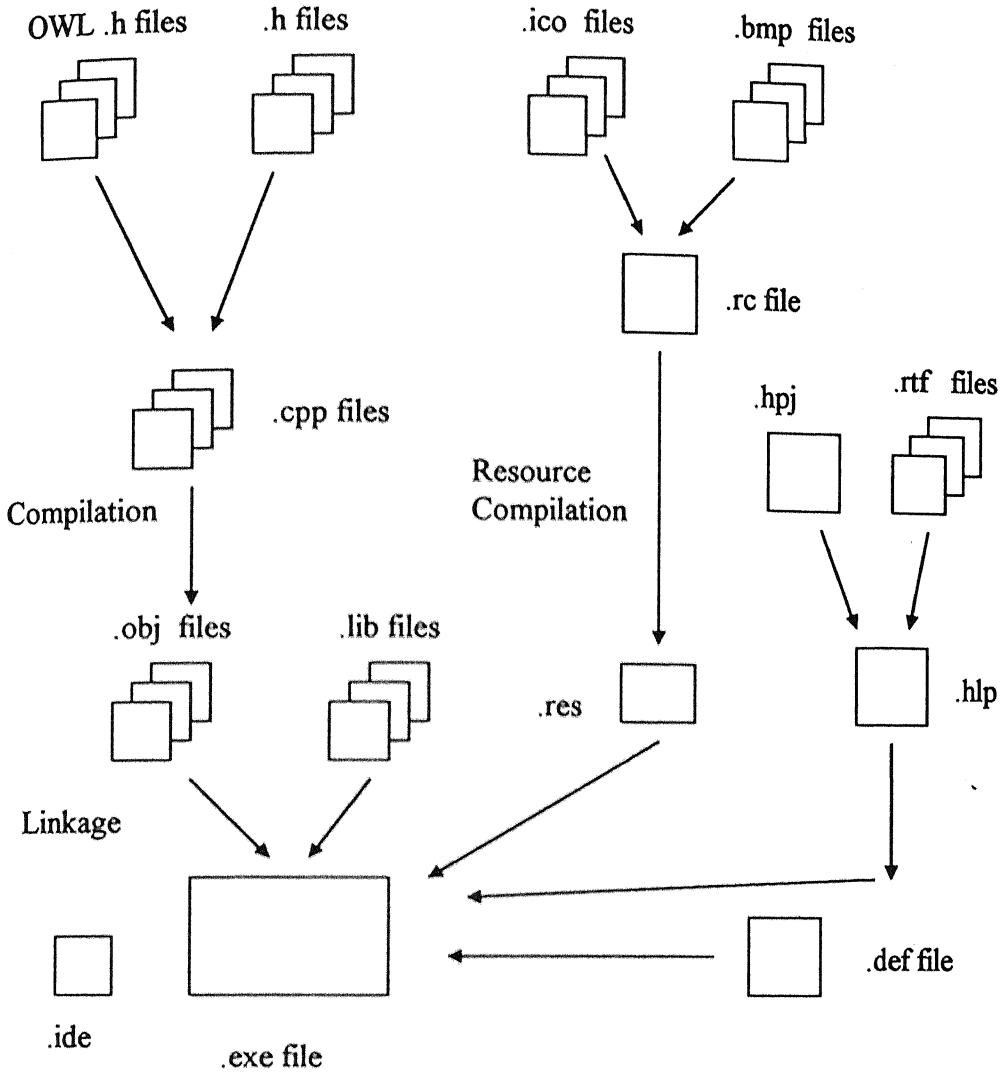


Figure D.1 : The file structure for the Application